Exhibit 11

Exhibit 11

AT&T

AT&T Wireless Services

White Paper

Developing Applications for the PocketNet Phone

Copyright Notice

Table of Contents

Developing Applications for the PocketNet Phone

This document describes the development environment for content providers that are planning to develop Internet- and Intranet-based applications for the PocketNet phone. This outline describes how to build applications, tools and skills required, and what administrative mechanisms should be in place to accommodate development. This document will be updated as tools and service offerings become available.

Architecture

The following description of the PocketNet network architecture will be used as a reference for this documentation.

Basic Components

A brief description of the basic architecture components follows.

CDPD Infrastructure

Network services offered by AT&T Wireless Data Division as outlined in the Plan of Record document version 1.4.

UP.Link Gateway

The required server that enables PocketNet or hand-held devices and services on the CDPD network.

PocketNet

Any Mobile-End System (M-ES) with a UP.View browser. The initial offering will support the PCSI and Mitsubishi phones or the Windows 95/NT phone simulator.

Content Provider

Anyone wishing to create applications or have applications created to enable information specifically for use with PocketNet devices. This information may come from existing Web-based services or database repositories.

PocketNet Web Service

A PocketNet Web service can be described as a set of URL's and arguments provided through any Web server's CGI input mechanism that returns compiled HDML as output.

PocketNet (Internet) Service Provider

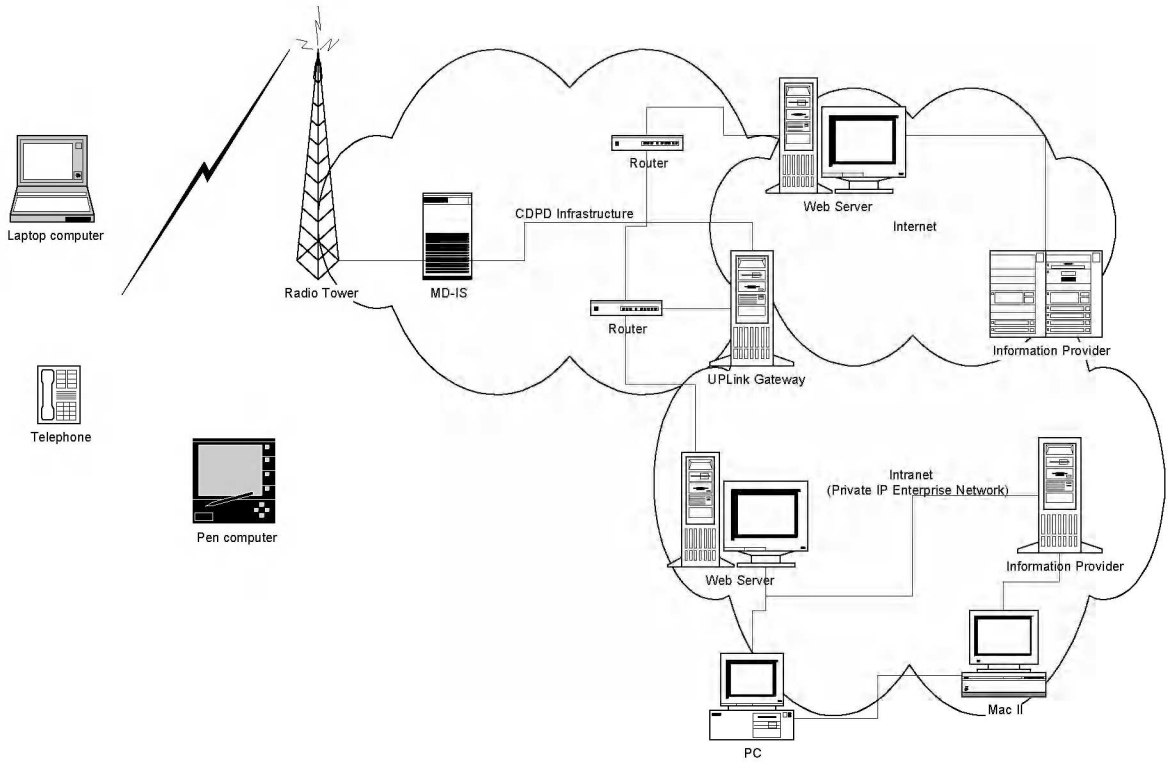
Any Internet or Intranet service provider that manages subscribers, Internet services, and an UP.Link Gateway. The PocketNet service provider must provide connectivity

to a CDPD network and is responsible for enabling PocketNet devices and content provider services.

Architecture Diagram

The following diagram depicts the basic components of a UP.Link-enabled environment:

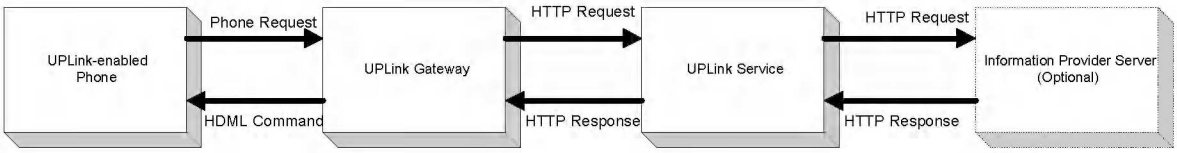
Figure 1. PocketNet Phone Architecture Diagram



How Does PocketNet Service Work?

The basic flow of information from a PocketNet can be described as a set of chained requests to an Information Web server as depicted in the diagram below.

Figure 2. PocketNet Information Flow

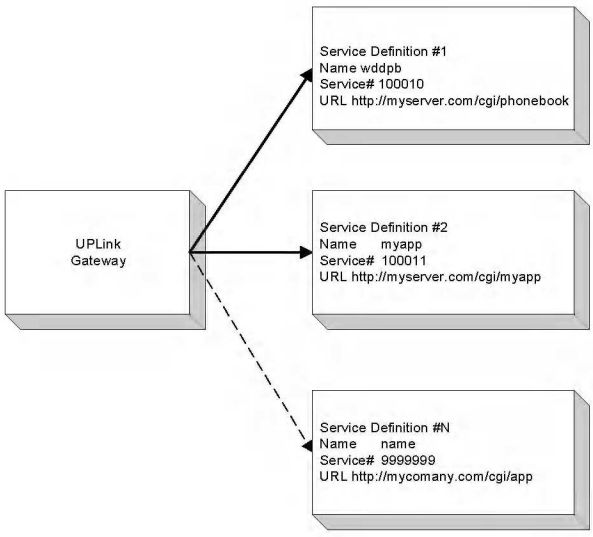


Registration Services

PocketNets and Virtual PocketNets

For a phone to work with an UP.Link Gateway, it must be registered in the Gateway's database.

Figure 3. UP.Link Gateway Registration Model



PocketNet Application Services

For a service to receive phone requests from an UP.Link Gateway, its URL must be registered in the Gateway database.

What Do I Need to Know to Develop Internet Applications?

Developers should have the following experience and knowledge base in order to successfully develop applications for the PocketNet phone:

- TCP/IP networking
- Basic sockets programming is helpful
- Windows NT or UNIX programming environment
- How a HTTPD (Web) server works
- HyperText Markup Language (HTML)
- Common Gateway Interface (CGI)

CGI Reference	URL
CGI Specification	http://www.ast.cam.ac.uk/%7Edrtr/cgi-spec.html-
Usenet - Writing CGI scripts for the Web	comp.infosystems.www.authoring.cgi-
A CGI Programmer's Reference	http://www.best.com/~hedlund/cgi-faq/-
Common Gateway Interface	http://hoohoo.uiuc.ncsa.edu/cgi/interface.html-

- Universal Resource Locator (URL) Parameter Passing Specification
- The C, C++ and/or Perl programming languages

Perl Web Reference	URL
Perl.com	http://www.perl.com-
Perl Basics	http://briet.berkeley.edu/perl/perl_tutorial.txt-
Perl Manual in HTML	http://www.cis.ufl.edu/cgi-bin/plindex-
Perl Reference Materials	http://www.eecs.nwu.edu/perl/perl.html-

What Will I Need to Learn?

Handheld Device Markup Language (HDML)

HDML is a relatively simple Standard Generalized Markup Language (SGML) DTD similar to HTML. It consists of angle bracket-delimited statements that define interactions between a user and an UP.Link-enabled phone.

Like HTML, HDML uses the ASCII character set. The HDML compiler ignores the case of HDML keywords. It also converts one or more contiguous new lines, carriage returns, tabs, or spaces to a single space. HDML examples in this manual are formatted with new lines and tabs to make them easier to read. However, this formatting is not required for HDML to be valid. In fact, it is removed by the HDML compiler.

How to Retrieve Web Pages From Other Web Servers

The primary method of retrieving information for a UP service requires the application to call out URL functions of other Web servers. Functions to assist this task can be found in the SDK libraries.

What's in the Unwired Planet SDK?

The UP.Link Software Developer's Kit

This free UP.Link software developer's kit (SDK) provides tools for developers who wish to create or maintain PocketNet application services. It includes the following components and is available on UP's Web site at *http://www.uplanet.com*.

The UP.Link phone simulator

The simulator is for Windows 95 and Windows NT. It simulates the behavior of an UP.Link-enabled phone and makes it easy to test UP.Link services.

Perl or C++ function libraries

These function libraries simplify the process of generating HTML and HTTP requests. The UP.Link developer's kit provides Perl utilities that make it easy to connect to HTTP sites and retrieve Web pages.

The following table lists these utilities and their use:

UP SDK Function	Purpose
HTTPConnect()	Connects a specified URL to a socket.
HTTPDisconnect()	Disconnects the current URL from a specified socket.
HTTPSendRequest()	Requests a web page from a specified socket.
URLConnect()	Connects a socket to the specified port or service of a specified host.
URLParse()	Parses a URL into a service protocol, a hostname, and a filepath.

The HDML compiler

The HDML compiler compiles HDML into a compact format that the UP.Link Gateway can interpret.

UP.Link developer documentation

Two online manuals are provided in Adobe Acrobat (PDF) format. These manuals are complete descriptions of how to develop applications and also serve as reference manuals.

Creating a PocketNet Application

The following examples from the UP.Link SDK explain how a developer creates an application.

Perl Example: Retrieving Information from Another Web Site

The Target Web Site

Suppose you want to provide a wave condition service for surfers. Florida State University provides Web pages with continuously updated, real-time wave information from National Oceanographic and Atmospheric Administration (NOAA) marine buoys. For example, the Web page for the Santa Cruz, California, buoy contains HTML code similar to the following:

```
<html>
<head>
<META HTTP-EQUIV="Refresh" Content=360>
<title>Current Observation for Station 46042</title>
. . .
<h2>Oceanographic Data</h2>
Sea Surface Temperature: 53.1 F<br>
Wave Height: 8.5 ft.<br>
. . .
```

The Perl Code

The following Perl code retrieves this Web page, reformats its data into HDML, and generates an HTTP response. Following the code is a line-by-line explanation of how it works.

```
1 #!/usr/local/bin/perl5.001
2 #
3 require "HTTPClient.pl"; #Utils provided with Developer's Kit
4
5       &HTTPClient'HTTPConnect(*HTTP,
6               "http://www.met.fsu.edu/nws/cgi-bin/buoy.cgi?46042");
7       while(<HTTP>)
8       {
9               if(/Wave Height/i)
10              {
11                      $height = $_;
12                      last;
13              }
14      }
15      if($height eq undef)
16      {
17              $height = "Wave height not available";
18      }
19      &HTTPClient'HTTPDisconnect(*HTTP);
20      $DECK = "<HDML VERSION=0.1 TTL=0>\n".
21      "\t<DISPLAY>\n".
22      "\t\t<CENTER>Santa Cruz\n".
23      "\t\t<WRAP>$height".
24      "\t</DISPLAY>\n".
25      "</HDML>\n";
26
27      print "Content-type: application/x-til\n\n";
28
29      open(HDMLOUT, "|./hdml2til");
30      select HDMLOUT;
31      print $DECK;
```

The Explanation

The following table presents a line-by-line description of the Perl code in the previous section.

Line 1	This line identifies the Perl version to use. Change this line as needed for your server.
Line 3	This line includes the HTTPClient.pl utilities file provided by Unwired Planet. This file defines the HTTPConnect() and HTTPDisconnect() functions, which the application uses to retrieve Web pages.
Lines 5-6	These lines use the HTTPConnect() utility function to connect a socket to the Web page containing the wave data. The function reads the HTTP status line, but leaves the rest of the output.
Lines 7-18	These lines search the HTML of the retrieved Web page for the line containing the wave height information (the line starting with the string, Wave Height). If the line isn't found, an error message is stored to the variable that gets displayed on the phone.
Line 19	This line uses the HTTPConnect() utility function to disconnect the socket from the Web page.
Lines 20	This line defines the header of the HDML deck to be generated. The VERSION option in the <HDML> statement specifies the HDML version the deck uses (0.1). The TTL option specifies how long, in seconds, the phone should cache the deck. Because the wave information is extremely timely, the option specifies 0, which means the phone doesn't cache the deck at all. Note that the tabs (\t) and the linefeeds (\n) characters do not affect the text displayed on the phone. These are included only to format the generated HDML so that it is easy to read.
Lines 21-24	These lines define a display card that displays the string Santa Cruz and the line from the Web page that provides the wave height. The <CENTER> tag instructs the phone to center the string Santa Cruz. The <WRAP> tag instructs the phone to wrap the string that provides the wave height on to the next line if it exceeds the display width.
Line 27	Prints the HTTP header. The header specifies the content type of the HTTP response (compiled HDML, or til).
Lines 29-30	These lines set standard output so that it is piped through the HDML compiler (hdm2til). If you pipe invalid HDML code to the HDML compiler, the compiler generates error messages. Note that this sample code omits error checking for the sake of simplicity. When you create a real service, it should check the HDML compiler output for errors.
Line 31	This line prints the HDML to standard output. Because the code pipes standard output through the HDML compiler, the actual output is compiled HDML.

C++ Example: Generating Compiled HDML

For every reply to a PocketNet phone, the application must compile the HDML cards for the UP.Link Gateway to properly deliver the reply. The following C++ example compiles HDML before sending it to the PocketNet.

The Task

The following code reads HDML from standard input and generates compiled HDML to standard output. Following the code is a line-by-line explanation of how it works.

```
1 #include <iostream.h>
2 #include "hdmlc.h"
3 4 int main()
5 {
6       TpcParser parser("Standard Input", cin);
7       if (parser.Parse())
8       {
9               TpcHDMLCGen gen(cout);
10              parser.Generate(&gen);
11      }
12      else
13      {
14              TpcParseError *err;
15              err = parser.GetErrors();
16              while (err)
17              {
18                      cerr << err << endl;
19                      err = err->GetNext();
20              }
21 }
```

The Explanation

The following table presents a line-by-line description of the C++ code in the previous section.

Line 2	Includes the header file containing the HDML compiler classes.
Line 6	Creates a parser object. The first argument defines the name of the file from which to read the HDML. The second argument instructs the TpcParser object to read from the cin iostream (standard input).
Line 7	Instructs the parser object to parse the input. If it succeeds, it returns TRUE; otherwise, it returns FALSE.
Lines 8-11	Output the HDML if the parser succeeds in parsing the HDML input into an AST. Line 8 creates a TpcHDMLCGen generator object and set its output to the standard output iostream (cout). Line 10 instructs the parser to generate output. The TpcParser::Generate() method scans the AST, asking the generator object to output the AST in compiled HDML.
Lines 13-20	Output error messages if the parser fails to parse the HDML into an AST. Line 14 declares a pointer to an error object. The error class is a linked list of errors. Line 15 uses TpcParser::GetErrors() method to get the head of error list. Lines 17-20 traverse the list of errors, printing each error. Each TpcParserError object provides an interface to write itself to the iostream.